Pergamon

Computers Math. Applic. Vol. 30, No. 2, pp. 27–36, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00075-5

The Temporal Boolean Derivative Applied to Verification of Extended Finite State Machines

E. VANDERMEULEN
LGI2P, EMA-EERIE Parc Scientifique G. Besse, 30000 Nimes, France

H. A. DONEGAN
University of Ulster, Jordanstown, County Antrim
BT37 OQB, Northern Ireland

M. LARNAC AND J. MAGNIER
LGI2P, EMA-EERIE Parc Scientifique G. Besse, 30000 Nimes, France

(Received December 1994; accepted January 1995)

Abstract—Extended finite state machines are an important feature of modern computers. Their verification, unlike sequential system testing, is very complex and has received little attention in literature. This paper suggests a model based on a symbolic representation to describe the temporal behavior of sequential machines. Two examples of different architectures illustrate the application of the methodology.

Keywords—Boolean derivative, Temporal logic, Sequential machine.

INTRODUCTION

Sequential machines demand quality production coupled with optimum testing procedures. Such procedures attempt to verify that the device under test possesses the functional characteristics that it was intended to have. This procedure is known as *functional testing* which would, for example, verify that a combinational circuit behaves as its truth table demands, and moreover, verify that a sequential system operates according to its state transition and output tables. The primary purpose of functional testing is to determine if design mistakes exist within the device.

The method of this research is based on the *extended finite state machine* (*efsm*), a generalization of the traditional *finite state machine* (*fsm*) model [1]. An *efsm* can be represented using an appropriate formal system. In this particular case, temporal logic is used to represent their behavior. The logic is that described in [2,3].

The output values of a *fsm* at a given time depend not only on the present inputs, but also on previously applied inputs. Essentially, the history of previous inputs is summarized in the state of the system. *Efsm* verification, with which we are primarily concerned, is considerably more difficult than combinational logic verification, and has received much less attention. To apply functional testing to sequential machines, the procedures are three-fold. First, symbolic

E. Vandermeulen wishes to thank J.G. Hughes and D.A. Bell of the Faculty of Informatics at the University of Ulster, Northern Ireland for arranging accommodations and facilities to enable the preparation of this paper during the summer of 1994. Without the collaborative efforts of D. Pearson at Ecole des Mines, the opportunity would not have arisen.

representation is used to map the behavior of the *efsm* to a list of formulae called *elementary valid formulae* (*evf*). Second, more abstract reasoning is required so that conditions involving the same event can be represented within *unified valid formulae* (*uvf*). The precise meanings of *evf* and *uvf* are described later in the paper. Finally, the Boolean derivative is brought into play. This is used to calculate the sensitivity of any *uvf* with respect to a given variable. All faults on the present and next state transition are considered.

Boolean derivatives are used in algebraic techniques to develop test patterns for combinational circuits. This is essentially a method of determining the primary inputs required to force a function to be sensitive to a particular input variable. It is very interesting from an analytical point of view, but has limited practicality, because it requires that a *switching function* be developed to describe the combinational circuit under consideration. Moreover, the Boolean derivative is computationally intensive for implementation on a digital computer.

The testing strategy is applied to logic circuits which comprise each device. These circuits are constructed by interconnecting elements called *gates* whose inputs and outputs are characterized by the binary digits 0 or 1, the output of each gate which can, for example, be an AND gate or an OR gate can be represented by a logic or a Boolean function of the inputs.

Akers *et al.* [4] have proposed an efficient method to compute Boolean functions using *binary decision diagrams* (*bdd*). The limitation of this method is based on the size of the Boolean functions that a *bdd* can represent. It is assumed that the behavior of the circuit is completely described.

DEFINITIONS

Finite State Machines

Our attention will be primarily focused on deterministic synchronous completely specified machines, which possess the property that the next state is determined uniquely by the present state and the present input. A typical sequential machine is shown in Figure 1. It consists of a combinational logic block and feedback latches that hold the information. It is assumed that the present state and the next state are neither directly controllable nor observable. Usually, a sequential machine is represented by a quintuple, $(\mathcal{S}, \mathcal{X}, \mathcal{Z}, \delta, \lambda)$ [5], where $\mathcal{S}$ is a nonempty set of states, $\mathcal{X}$ a nonempty set of inputs, $\mathcal{Z}$ a nonempty set of outputs, where

and $\delta : \mathcal{X} \times \mathcal{S} \rightarrow \mathcal{S}$ is the state transition function,
and $\lambda : \mathcal{X} \times \mathcal{S} \rightarrow \mathcal{Z}$ is the output function.

A state is a bit pattern of length equal to the number of memory elements (latches or flip-flops) in the sequential circuit, just as input and output are bit patterns of lengths corresponding to the number of input and output bits of the circuit. They each represent a combinational value of Boolean variables. Each state, input, and output pattern comprises components which correspond, respectively, to sets of flip-flops, input values, and output values.

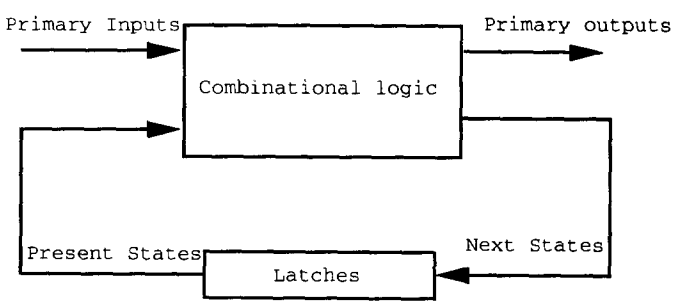

Figure 1. Finite state machine.

Figure 2 illustrates an example of a *fsm* with one input (e), two flip-flops (A,B), two gates and one output (z). This machine is controlled by a clock. The flip-flop characteristics and output equation of Figure 2, where A+ and B+ are next values of respective flip-flops, are shown in the following equations.

$$A+ = e.A' + (e' + B).A,$$
$$B+ = e.B' + (e.A' + e'.A).B,$$
$$z = B$$

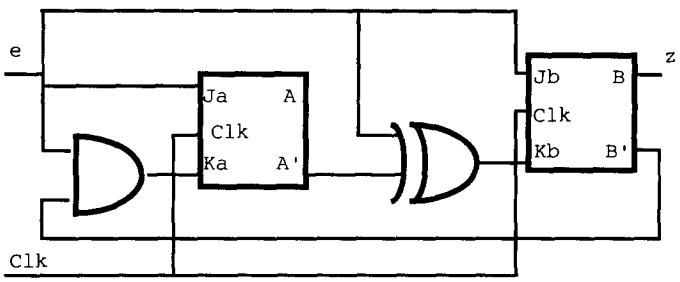

Figure 2. Example of a finite state machine.

Extended Finite State Machine

An Extended Finite State Machine E is defined [1] as the 7-tuple $(\mathcal{S}, \mathcal{I}, \mathcal{O}, \mathcal{D}, \mathcal{F}, \mathcal{U}, \mathcal{T})$, where:

$\mathcal{S}$ is a nonempty set of symbolic states,
$\mathcal{I}$ is a nonempty set of input symbols,
$\mathcal{O}$ is a nonempty set of output symbols,
$\mathcal{D}$ is an n-dimensional space,
$\mathcal{F}$ is a nonempty set of enabling functions f_i such that $f_i : \mathcal{D} \rightarrow \{0, 1\}$,
$\mathcal{U}$ is a nonempty set of update transformations U_i such that $U_i : \mathcal{D} \rightarrow \mathcal{D}$,
$\mathcal{T}$ is a transition relation such that $\mathcal{T} : \mathcal{S} \times \mathcal{F} \times \mathcal{I} \rightarrow \mathcal{S} \times \mathcal{U} \times \mathcal{O}$.

δ is used to denote an n-dimensional vector with the components $\delta_i \in \mathcal{D}_i$ and the transition $\mathcal{T}((S_1, f, I), (S_2, U, O))$ is denoted by $(S_1, f, I) \rightarrow (S_2, U, O)$ where $S_1, S_2 \in \mathcal{S}, f \in \mathcal{F}, U \in \mathcal{U}$, and $O \in \mathcal{O}$. Further, $(S_1, f, I) \rightarrow (S_2, U, O$, means that if E is in symbolic state S_1 with a vector of variables δ such that $f(\delta) = 1$ and the input I is received, then E moves to the symbolic state S_2, while generating output O and performing the update $\delta \leftarrow U_i(\delta)$.

The input and output sets of an *efsm* are partitioned and the extraction of the *fsm* shows two parts (see Figure 3). One is the *sequential machine* which is called the *control part* and the other is referred to as the *operational part.* The control part is an *fsm* as discussed above, and the operational part is composed of registers and functions. A system which is independent of the number of bits used to represent inputs and outputs is required. Both of the parts are synchronized using enabling and updating functions.

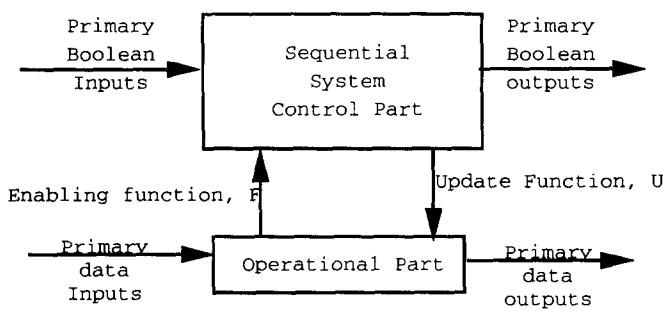

Figure 3. Extended finite state machine.

Example of an Extended Finite State Machine

This example is based on the work of Cheng and Krishnakumar [1]. Consider a machine M of 2 inputs i_1, i_2 and one register r_1 where i_1, i_2 and r_1 are, respectively, 1 bit, 8 bits, and 8 bits wide. The function of the circuit is described as follows: when i_1 is high, read in the value at i_2 and store it in r_1. Once it is stored in r_1, check its value. If the value is less than 16, increment it by 1 at each cycle. Otherwise, output the value in r_1 and repeat. This system can be modelled by the following *efsm*. The machine consists of two symbolic states S_0 and S_1. There are five transitions:

$$\begin{aligned}
\mathcal{T}_1 &: (S_0, \text{TRUE}, i_1 == 1) \rightarrow (S_1, r_1 = i_2, \text{Out} = 0);\\
\mathcal{T}_2 &: (S_0, \text{TRUE}, i_1 == 0) \rightarrow (S_0, \text{NULL}, \text{Out} = 0);\\
\mathcal{T}_3 &: (S_1, r_1 < 16, i_1 == 1) \rightarrow (S_1, r_1 += 1, \text{Out} = 1);\\
\mathcal{T}_4 &: (S_1, r_1 \geq 17, i_1 == 1) \rightarrow (S_0, \text{NULL}, \text{Out} = r_1);\\
\mathcal{T}_5 &: (S_1, \text{TRUE}, i_1 == 0) \rightarrow (S_1, \text{NULL}, \text{Out} = 1).
\end{aligned}$$

The interpretation of $\mathcal{T}_3$ is: if the *efsm* is in state S_0 and the register r_1 is less than 16 and the input $i_1 == 1$, then the next state will be S_0 and the register r_1 is incremented by 1 and the output is 1.

We remark that, in this example, if we had used a *fsm* definition, we would have required additional transitions.

Linear Discrete Temporal Logic

Traditional propositional logic is extended by temporal operators, which allow the expression of time varying properties as they occur in the behavior of sequential systems. Here, we remind the reader of some basic elements of temporal logic as presented and used by Manna and Pnueli [2] and Magnier [3,6]. This technique was developed by these authors for specification and software verification.

Symbols: $\neg$, $\wedge$, $\vee$, $\supset$ are respectively read as: Not, And, Or, Involve.

Consider the situation where an *efsm* moves from a state to another state. The $\bullet$ operator represents the next instant that is associated with the next state and is called 1-future instant.

Using this definition

$\bullet^1 = \bullet$, represents the 1st future instant

and $\bullet^n = (\bullet^{n-1})\bullet$ represents the nth future instant.

Propositional temporal logic is an extension of the classical logic

$\bullet$ (next), $\Box$ (always).

Priority: $\neg, \bullet, \Box, \wedge, \vee, \supset$.

Boolean Derivatives

The Boolean derivative of a function $f(v_1, \ldots, v_n)$ with respect to the variable v_i is defined as

$$\frac{\partial f}{\partial v_i} = f(v_1, \ldots, v_{i-1}, 0, v_{i+1}, \ldots, v_n) \oplus f(v_1, \ldots, v_{i-1}, 1, v_{i+1}, \ldots, v_n),$$

where $f(v_1, \ldots, v_{i-1}, 0, v_{i+1}, \ldots, v_n)$ is the function f evaluated with v_i being 0, $f(v_1, \ldots, v_{i-1}, 1, v_{i+1}, \ldots, v_n)$ is the function f evaluated with v_i being 1, and $\oplus$ is the exclusive-or operator. If $\frac{\partial f}{\partial v_i}$ is 0, the function f is completely independent of the variable v_i, that is, the value of f does not change when the value v_i changes. If $\frac{\partial f}{\partial v_i}$ is 1, the function f depends directly on v_i [3,5,7].

Here are some properties [3,5]

$$\frac{\partial f(v_1, \ldots, v_n)}{\partial v_i} = \frac{\partial \neg f(v_1, \ldots, v_n)}{\partial v_i},$$

$$\frac{\partial f(v_1,\ldots,v_n)}{\partial v_i} = \frac{\partial f(v_1,\ldots,v_n)}{\partial \neg v_i}.$$

Boolean derivatives are used to determine test patterns for faults in sequential systems and to verify properties of *fsm*. The most common fault model is the logical stuck-fault model [5,7] in which there are three basic assumptions:

(1) a fault results in a module responding as if one of its inputs or outputs is physically stuck at 1 or 0,
(2) the basic functionality of the circuit is not altered by the fault, and
(3) the fault is permanent.

The logic module can be a single gate or a collection of gates that implement some logic function.

The use of the Boolean derivative to determine test patterns for faults that occur on the primary outputs consists of two fundamental steps. First, if a test pattern is being sought for primary input v_i *stuck-at-1* fault, v_i must be selected as 0 to attempt to force the line (signal) to deviate from its faulty value. Likewise, if a test pattern is being sought for v_i *stuck-at-0* fault, select v_i as 1. Second, select the remaining primary inputs such that the output is sensitive to the value of v_i. This second step is accomplished by forcing the Boolean derivative with respect to v_i to be 1. These two steps can be placed in equation form as:

$$\neg v_i \wedge \frac{\partial f}{\partial v_i} = 1 \text{ for stuck-at-1 faults} \quad \text{and} \quad v_i \wedge \frac{\partial f}{\partial v_i} = 1 \text{ for stuck-at-0 faults.}$$

EFSM BEHAVIOR IN TEMPORAL LOGIC

Elementary Valid Formulae

This research suggests a method based on a temporal logic algebra [3] which describes the temporal behavior of *efsm*. This algebra allows us to calculate the temporal Boolean derivative of a formula with respect to an event. In [3], an *evf* is equivalent to a transition of a *fsm*. We have to extend this definition to the *efsm*.

The principal advantage of an *efsm*, due to Cheng [1], is the capability of operating with a set of states and sets of transitions simultaneously. This advantage is embodied in the set of enabling functions whereby the register variables, primary inputs and primary outputs are integrated for the utilisation of the update functions. The remaining problem asks how to denote the data inputs and data outputs from the operational part.

The method uses symbolic representations and the reader is reminded that states, inputs and outputs were described as patterns. For each transition, it is desirable to have a functional formula such that when the *fsm* is in state S_1 with input X, the *fsm* moves to state S_2 with the output Z.

For example, referring to Figure 2, S_0 is the state of the *fsm* when the value of $A = 0$ and the value of $B = 0$, back so-on with the other states, inputs and outputs.

Table 1. Variable change.

\	A	B
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

\	e
X_0	0
X_1	1

\	B
Z_0	0
Z_1	1

Henceforth, we use Boolean variables which are true either if the *fsm* is in state S or if the input is X and the output is Z. When a variable is false, it means that almost only one of the remaining variables can be true. For example, s_0 True means that the *fsm* is in state S_0, whereas $\neg s_0$ means

that the *fsm* is in any other state except S_0. To represent this functional behavior in temporal logic, *elementary valid formulae* (*evf*) [3] are used. In this case:

$$s_0 \wedge x_0 \supset \bullet s_0 \wedge z_0,$$

$$s_0 \wedge x_1 \supset \bullet s_3 \wedge z_0,$$

$$s_1 \wedge x_0 \supset \bullet s_0 \wedge z_1,$$

$$s_1 \wedge x_1 \supset \bullet s_3 \wedge z_1,$$

$$s_2 \wedge x_0 \supset \bullet s_2 \wedge z_0,$$

$$s_2 \wedge x_1 \supset \bullet s_1 \wedge z_0,$$

$$s_3 \wedge x_0 \supset \bullet s_3 \wedge z_1,$$

$$s_3 \wedge x_1 \supset \bullet s_2 \wedge z_1,$$

where

- s_i is the Boolean variable that returns True if the *fsm* is in state S_i, and returns False if the *fsm* is in any other state except S_i.
- x_i is the primary Boolean variable that returns True if the input to the *fsm* is X_i, and returns False if the input to the *fsm* is any other input except X_i.
- z_i is the primary Boolean variable that returns True if the output from the *fsm* is Z_i, and returns False if the output of the *fsm* is any other output except Z_i.

Note that for n inputs, m flip-flops, and p outputs, this method requires 2^n symbolic input patterns, 2^m symbolic state patterns, and 2^p symbolic output patterns, all of which can be represented by binary decisions diagrams [8].

To correspond with the *fsm* symbolic representation, the following variable change is made:

X denote a set of vectors that represent inputs and
Z denote a set of vectors that represent outputs.

With such a variable change, we can represent the input of the *efsm*, for example $i_1 = 0$, by a symbolic input X_0. In other words, x_0 is true when 0 is the value of the input i_1.

For the purpose of this research, functions are denoted as Boolean variables, thus:

f_i is a function that returns a Boolean value.
u_i is a Boolean variable that takes the value True when the update function U_i is running, and False if any other update function is running except U_i.

Now, it is possible to represent an *efsm* in temporal logic formalism. Let an *evf* be defined as follows:

$$evf_{p,p'} = \Box\left[s_p \wedge f \wedge x \supset \bullet s_{p'} \wedge u \wedge z\right],$$

where s, f, x, u, z are propositional variables associated with each Boolean variable indicating, respectively, state, function, input, update function, and output. An example of the *evf* in this case is as follows:

$$s_0 \wedge f_1 \wedge x_1 \supset \bullet s_1 \wedge u_3 \wedge z_0,$$
$$s_0 \wedge f_1 \wedge x_0 \supset \bullet s_0 \wedge u_1 \wedge z_0,$$
$$s_1 \wedge f_2 \wedge x_1 \supset \bullet s_1 \wedge u_2 \wedge z_1,$$
$$s_1 \wedge f_3 \wedge x_1 \supset \bullet s_0 \wedge u_1 \wedge z_1,$$
$$s_1 \wedge f_1 \wedge x_0 \supset \bullet s_1 \wedge u_1 \wedge z_1.$$

The interpretation of the third one is: if the state s_1 is true, and if the enabling function f_2 is true, and the input is x_1, then the next state will be s_1, and the update function is u_2, and the output is z_2.

Unified Valid Formulae

Analysis of the behavior of *efsm* requires a more global approach so that conditions involving future events can be obtained. This motivates the definitions of the *uvf* which enables the consideration of all terms involved in the future event. Let an *uvf* be defined as follows:

$$Et = \bullet sp \text{ or } Et = u \text{ or } Et = z:$$

$$uvf(Et) = \Box \left[\bigvee_{(p,q,r):s_p \wedge f_q \wedge x_r \supset Et} [s_p \wedge f_q \wedge x_r] \right].$$

Example of *fsm uvf* in this case:

$$\begin{aligned}
uvf(\bullet s_0) &= (s_0 \wedge x_1) \vee (s_1 \wedge x_0),\\
uvf(\bullet s_1) &= s_2 \wedge x_1,\\
uvf(\bullet s_2) &= (s_2 \wedge x_0) \vee (s_3 \wedge x_1),\\
uvf(\bullet s_3) &= (s_0 \wedge x_1) \vee (s_1 \wedge x_1) \vee (s_3 \wedge x_0),\\
uvf(z_0) &= (s_0 \wedge x_0) \vee (s_0 \wedge x_1) \vee (s_2 \wedge x_0) \vee (s_2 \wedge x_1),\\
uvf(z_1) &= s_2 \wedge x_1.
\end{aligned}$$

Example of *efsm uvf*:

$$\begin{aligned}
uvf(u_1) &= (s_0 \wedge f_2 \wedge x_0) \vee (s_1 \wedge f_2 \wedge x_1) \vee (s_1 \wedge f_1 \wedge x_0),\\
uvf(u_2) &= s_1 \wedge f_2 \wedge x_1,\\
uvf(u_3) &= s_0 \wedge f_1 \wedge x_1,\\
uvf(\bullet s_1) &= (s_0 \wedge f_1 \wedge x_1) \vee (s_1 \wedge f_2 \wedge x_1) \vee (s_1 \wedge f_1 \wedge x_0),\\
uvf(\bullet s_0) &= (s_0 \wedge f_1 \wedge x_0) \vee (s_1 \wedge f_3 \wedge x_1),\\
uvf(z_0) &= (s_0 \wedge f_1 \wedge x_1) \vee (s_0 \wedge f_1 \wedge x_0),\\
uvf(z_1) &= (s_1 \wedge f_2 \wedge x_0) \vee (s_1 \wedge f_1 \wedge x_0),\\
uvf(z_2) &= s_1 \wedge f_3 \wedge x_0.
\end{aligned}$$

For example, $uvf(u_1)$ gives all the conditions that involve u_1. They are:

- The *efsm* state is S_0 and the input is $X_0(i_1 = 0)$ the result of the enabling function f_2 is True or
- The *efsm* state is S_1 and the input is $X_1(i_1 = 1)$ and the result of the enabling function f_2 is True or
- The *efsm* state is S_1 and the input is $X_0(i_1 = 0)$ and the result of the enabling function f_1 is True.

The same interpretation can be written for states *uvf* and outputs *uvf*. One use of these *uvf* is to give some properties of *efsm*. For example, examining $uvf(\bullet^n s_1)$ reveals sequences called synchronizing sequences that move the *efsm* into a reset state. In the case of a completely specified machine, these formulae constitute the necessary and sufficient conditions which characterise a given event [3,6].

It is easy to prove that

$$\begin{aligned}
uvf(\bullet^2 s_0) = {} & s_0 \wedge [[(f_1 \wedge x_0) \wedge \bullet(f_1 \wedge x_0)] \vee [(f_1 \wedge x_1) \wedge \bullet(f_3 \wedge x_1)]] \vee\\
& s_1 \wedge [[(f_3 \wedge x_1) \wedge \bullet(f_1 \wedge x_0)] \vee [(f_2 \wedge x_1) \wedge \bullet(f_3 \wedge x_1)] \vee [(f_1 \wedge x_0) \wedge \bullet(f_3 \wedge x_1)]].
\end{aligned}$$

Deterministic Rules

The definition of a deterministic machine reveals that it cannot be in two different states at any given moment. This property suggests the description of deterministic rules which simplify the method.

$\neg(\Box(s_i \wedge s_j))_{i\neq j}$ s_i and s_j can never be true at the same time.
$\neg(\Box(x_i \wedge x_j))_{i\neq j}$ x_i and x_j can never be true at the same time.
$\neg(\Box(z_i \wedge z_j))_{i\neq j}$ z_i and z_j can never be true at the same time.

These initial rules result from [3]. The same kind of rules apply with *efsm* and are extended with the update and enabling functions. As shown in the example above, more than one enabling function can be True at the same moment. This means that the enabling function (f) is associated with input (x) as a pair. Now, it is possible to write

$\neg(\Box(u_i \wedge u_j))_{i\neq j}$ u_i and u_j can never be true at the same time.
$\neg(\Box((f_i \wedge x_j) \wedge (f_l \wedge x_m)))_{(i\neq l)\vee(j\neq m)}$. Two different pairs can never be True at the same time.

THE TEMPORAL BOOLEAN DERIVATIVE

Boolean Derivative with Temporal Logic

Magnier [3,6] has proposed the principle of building a Boolean derivative using temporal logic. Let Et be a next state Boolean event or an update Boolean event and q some variable, representing either a state, an enabling function, or an input, ie,

$$q = s \text{ or } q = f \text{ or } q = x \text{ and } Et = \bullet s_p \text{ or } Et = u \text{ or } Et = z, \text{ then}$$

$$\frac{\partial Et}{\partial q} = (uvf(Et), q) \oplus (uvf(Et), \neg q).$$

To calculate this expression in temporal logic, it is necessary to represent $(uvf(Et), q)$ as follows:

$$\begin{aligned}
(uvf(Et), q) &= C_t(q) && : \text{the conditions that give } Et \text{ knowing that } q \text{ is True.}\\
(uvf(Et), \neg q) &= C_t(\neg q) && : \text{the conditions that give } Et \text{ knowing that } q \text{ is False.}\\
\neg(uvf(Et), q) &= \neg C_t(q) && : \text{the conditions that give } \neg Et \text{ knowing that } q \text{ is True.}\\
\neg(uvf(Et), \neg q) &= \neg C_t(\neg q) && : \text{the conditions that give } \neg Et \text{ knowing that } q \text{ is False.}
\end{aligned}$$

Hence,

$$\begin{aligned}
\frac{\partial Et}{\partial q} &= C_t(q) \oplus C_t(\neg q)\\
&= (C_t(q) \wedge \neg C_t(\neg q)) \vee (\neg C_t(q) \wedge C_t(\neg q)).
\end{aligned}$$

The result gives all the input sequences with their initial associated states which express the sensitivity of Et with respect to q. In this, the first part of the research $uvf(\bullet^n Et)$ is calculated for the n^{th} future instant and $\frac{\partial Et}{\partial q}$ is calculated only at 1-future instant.

For example, if q is the state s_{i_0} and Et as described above:

$$\frac{\partial Et}{\partial s_{i_0}} = \bigvee_{j,k\in JK(i_0)} \left[[f_j \wedge x_k] \wedge \left[\bigwedge_{i\in I(j,k)} \neg s_i \right] \wedge \bigwedge_{(i',j',k')\in IJK_3 Et(j,k)} [\neg s_{i'} \vee \neg f_{j'} \vee \neg x_{k'}] \right]$$
$$\vee \bigvee_{(i,j,k)\in IJK_4(Et)} \left[f_j \wedge x_k \wedge \bigwedge_{k'\neq k,k_0} \neg s_{i'} \right],$$

where

$$\begin{aligned}
JK(i_0) &= \{(j,k) : [s_{i_0} \wedge f_j \wedge x_k] \supset Et\},\\
I(j,k) &= \{i : [s_i \wedge f_j \wedge x_k] \supset Et \text{ where } i \neq i_0\},\\
IJK_1Et &= \{(i,j,k) : [s_i \wedge f_j \wedge x_k] \supset Et, i \neq i_0\},\\
IJK_2Et(j,k) &= \{(i',j',k') : (i',j',k') \in IJK_1Et - \{(i',j,k)\}\},\\
IJK_3(j,k) &= \{(i',j',k') : (i',j',k') \in IJK_2Et(j,k) - \{(i',j',k') : i' \in I(J,K)\}\},\\
IJK_4(Et) &= \{(i,j,k) \in IJK_1Et\} - \{(i,j,k) : (j,k) \in JK(i_0)\}.
\end{aligned}$$

In the same way, it is possible to calculate the sensitivity of an event with respect either to a function or an input.

Examples

EXAMPLE 1. In this example, we show how to calculate the sensitivity of an event with respect to a state.

$$\frac{\partial u_2}{\partial s_1} = (uvf(u_2), s_1) \oplus (uvf(u_2), \neg s_1) = f_2 \wedge x_1,$$

$$\begin{aligned}
JK(0) &= (2,1),\\
I(2,1) &= \emptyset,\\
IJK_1 &= \emptyset,\\
IJK_2(2,0) &= \emptyset,\\
IJK_4 &= \emptyset.
\end{aligned}$$

If the input is x_1, and the function f_2 is True, then

when s_1 becomes $\neg s_1$ then u_2 becomes $\neg u_2$, or $\neg u_2$ becomes u_2,
or when $\neg s_1$ becomes s_1 then u_2 becomes $\neg u_2$, or $\neg u_2$ becomes u_2.

EXAMPLE 2. In this example, we show how to calculate the sensitivity of an event with respect to an enabling function.

$$\frac{\partial u_2}{\partial f_2} = (uvf(u_2), f_2) \oplus (uvf(u_2), \neg f_2) = s_1 \wedge x_1.$$

If the machine is in state s_1, and the input is x_2, then

when u_2 becomes $\neg u_2$ then f_2 becomes $\neg f_2$, or $\neg f_2$ becomes f_2,
or when $\neg u_2$ becomes u_2 then f_2 becomes $\neg f_2$, or $\neg f_2$ becomes f_2.

CONCLUSION

The investigation, based on Boolean algebra and temporal logic, illustrates a verification strategy at the functional transition level for extended finite state machines. Essentially we have shown that the unified value formulae collects all the terms that involve a temporal event and at the same time some pertinent properties of sequential machines are revealed. Moreover, the temporal Boolean derivative returns the sensitivity conditions of a given event. These temporal logic formulae, constructed using temporal logic operators and symbolic variables are defined on the *efsm* representation model. Consequently, the strategy allowed us to address the following problems:

- Formal proof of the properties of an *efsm*.
- Verification of an *efsm* against a reference model.
- Formal generation of symbolic sequences for verification.
- Sequence generation for functional tests.

It is proposed to extend the research to include experiments which will measure the memory and CPU times for various input sequences.

REFERENCES

1. K.-T. Cheng and A.S. Krishnakumar, Automatic functional test generation using the extended finite state machine model, In *Proceedings 30th ACM/IEEE Design Automation Conference*, pp. 86–91, (1993).
2. Z. Manna and A. Pnueli, Verification of concurrent programs: A temporal proof system, Technical Report STAN-CS-83-967, Dept of Computer Science, Stanford University, (June 1983).
3. J. Magnier, Representation symbolique et verification formelle de machines sequentielles, Ph.D. Thesis, Universite Montpellier II, France, (1990).
4. S.B. Akers, Functional testing with binary decision diagrams, In *Proceedings 8th Annual IEEE on Fault Tolerant Computing*, pp. 75–82, (1978).
5. Kohavi, *Switching and Finite State Machine*, McGraw-Hill, (1978).
6. J. Magnier, D. Pearson and N. Giambiasi, The temporal Boolean derivative applied to verification of sequential machines, In *Proceedings of the European Simulation Symposium (ESS 94)*, pp. 313–319, Istanbul, (1994).
7. B.W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publishing Company, (1989).
8. K.S. Brace, R.D. Rudell and R.E. Bryant, Efficient implementation of bdd package, In *Proceedings 27th Design Automation Conference (DAC 90)*, pp. 40–45, (1990).